Reducing Energy Waste Due to Idle Network Devices

Khan, R., & Khan, S. U. (2017). Reducing Energy Waste Due to Idle Network Devices. *IEEE Potentials*, *36*(5), 37 - 45. https://doi.org/10.1109/MPOT.2017.2695298

Published in:
IEEE Potentials

Document Version:
Peer reviewed version

Reducing Energy Waste due to Idle Network Devices

Rafiullah Khan and Sarmad Ullah Khan

***Abstract*—Network devices always demand full time Internet connectivity for remote access, VoIP & Instant Messaging (IM) clients and other Internet based applications. Their built-in low power management features are usually disabled by users due to their incapability of maintaining network connectivity. The concept of Network Connectivity Proxy (NCP) has recently been proposed as an effective mechanism for reducing energy waste by impersonating presence of sleeping devices. However, proposed strategies in literature for proxying applications are either impractical for today's proprietary applications or limited to only open-source applications. This paper addresses challenges in proxying presence of sleeping devices in order to reduce their network energy waste. To overcome the limitations of NCP concept, this paper proposes an intelligent collaborative proxying scheme which is practically realizable. The proposed system enables network devices to sleep when idle without losing network presence. It can proxy basic networking protocols as well as ensures to run applications on only and only single user device (smartphone, tablet, laptop or office PC) that is actively used at that specific moment. Additionally, the proposed system provides features such as auto-discovery, zero-configuration and seamless communication between devices without requiring any network settings from user.**

***Index Terms*—Green networking, Network Connectivity Proxy, energy efficiency, application proxying.**

I. Introduction

The number of end-user network devices in the world increases at a rapid rate. Today, the most frequently used devices include smartphones, tablets, laptops and desktop PCs. The Environmental Protection Agency (EPA) investigated energy usage of PCs in United States and revealed that they consume about 2% of overall generated electricity every year. Further, Lawrence Berkeley National Laboratory revealed in a study that about 60% people never switch OFF computers in offices (even when idle) [1]. The main reason behind such behavior is the need of full time availability (powered-up state) and Internet connectivity for remote access, VoIP & Instant Messaging (IM) clients and other Internet based applications [2]. It is worth to mention that PCs support low power states with almost 10x lower power consumption than idle state. However, such states are normally disabled by users due to their incapability of maintaining network connectivity. Thus, huge amount of energy is wasted to keep PCs powered-up 24/7 even when idle [3].

The concept of Network Connectivity Proxy (NCP) has recently been proposed as an effective mechanism for reducing energy waste by proxying presence of sleeping devices [1], [4]. Proxying presence (also known as impersonating presence) implies responding to packets on behalf of sleeping devices in order to make them appear as connected/online. The concept of NCP is illustrated in Fig. 1. In short, basic NCP functionalities on behalf of sleeping devices at different network layers include:

1) Maintaining link layer presence and reachability i.e., responding to Address Resolution Protocol (ARP) packets on behalf of sleeping devices. This is also known as ARP spoofing in which NCP associates its MAC address with the IP addresses of sleeping devices. Note: NCP also broadcasts gratuitous ARP reply packets (i.e., no ARP request received) whenever a client device enters into sleep state. This is also known as link layer traffic diversion and is useful to get access to packets intended for sleeping devices.
2) Maintaining network layer presence and reachability i.e., keeping IP addresses of sleeping devices alive. This requires NCP to respond to PING packets on behalf of sleeping devices. If a sleeping device is using Dynamic Host Configuration Protocol (DHCP), NCP needs to periodically renew its IP address lease with the DHCP server.
3) Maintaining transport layer presence and reachability i.e., proxying TCP sessions (which were active on client devices before sleeping) and allowing new TCP connections

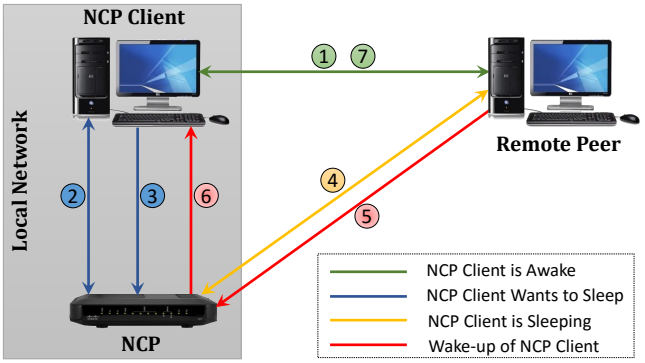

Figure 1. The NCP concept: (1) NCP client device is awake and communicating itself with remote peer, (2) NCP client registers with NCP and specifies/activates supported proxying rules, (3) NCP client informs NCP that it is going to sleep state, (4) NCP impersonates presence of sleeping client device with its remote peer, (5) NCP received a packet that requires to wake-up its client device (e.g., remote access request, new email or message received, call received on a VoIP application, etc), (6) NCP uses network-based wake-up technology (e.g., wake-on-LAN) to wake up its client device, (7) NCP client is once again fully powered ON and communicating itself with remote peer.

R. Khan is with school of Electronics, Electrical Engineering and Computer Science (EEECS) at Queen's University Belfast, Belfast, United Kingdom, Email:rafiullah.khan@qub.ac.uk

S.U. Khan is with department of Electrical Engineering at CECOS University of IT and Emerging Sciences, Peshawar, Pakistan, Email: sarmad@cecos.edu.pk

to be established by responding to TCP SYN packets.
4) Proxying application layer presence. Internet based applications periodically send and receive specific presence messages (also known as heartbeat messages). An application is considered disconnected if it fails to do so. NCP needs to generate/respond to these application-specific heartbeat messages.

The NCP wakes-up a sleeping device whenever necessary e.g., remote access packet received, new email received, new message received on IM application, call received on VoIP application, etc. To this aim, NCP implements network-based wake-up technology known as Wake-On-LAN (WOL). All modern Network Interface Cards (NICs) have an auxiliary power connection from the motherboard and remain powered up even when the devices themselves are sleeping. They also have connection to wake-up interrupt line on motherboard. To wake-up a sleeping device, the NCP broadcasts a WOL packet in the local network. The sleeping device's NIC detects it and triggers the device wake-up. The WOL packet contains in its payload 6 bytes of 1's followed by the MAC address of sleeping device's NIC repeated 16 times. Since detection of WOL packet is based on its payload content, it is not parsed by full protocol stack. It is normally sent over Ethernet (using EtherType 0x0842) or over UDP transport protocol (using port 9).

Apparently, the term NCP means network proxying but its scope encompasses application proxying as well [1]. A survey in [2] revealed that 52 % people in office environment leave PCs powered-ON 24/7 for remote access and 35 % do it for background applications (out of which IM applications constitute 47 %). Today, numerous remote access protocols and applications are available such as SSH (TCP port 22), Telnet (TCP port 23), PCAnywhere (TCP/UDP port 5632), RealVNC (TCP port 5900), TightVNC (TCP ports 5500,5800 and 5900) Remote-Desktop (TCP/UDP port 3389) and many others. The most challenging task for NCP is proxying of ever increasing number of applications. To proxy a VoIP or IM application, the NCP needs to perform 6 important tasks: (i) acquire the present state of application from NCP client just before sleeping, (ii) acquire the present state of any TCP session associated with the application, (iii) keep the TCP session alive by generating or responding to TCP keep-alive messages, (iv) impersonate the application state with application server by generating or responding to periodic application-specific heartbeat messages, (v) wake-up NCP client whenever necessary (e.g., IM message or VoIP call received), (vi) return the application state and TCP session back to NCP client when it wakes-up. Steps i-ii are performed when NCP client is about to sleep. Steps iii-v are performed when NCP client is sleeping. Step vi is performed when NCP client wakes-up. To implement proxying feature for an application, knowledge of its source-code is required. Thus, NCP has been addressed until now only for open-source applications such as xChat and KaduChat [5], ICQ [2], Jabber [6], etc. However, most commonly used applications in daily life are closed-source (such as WhatsApp, Viber, Skype, imo, ooVoo, WeChat, etc) for which proxying capability cannot be implemented. Depending on NCP capabilities, it processes each packet on behalf of sleeping devices by taking one of the following actions:

1) Ignore the packet if it is a broadcast, multicast, routing or port scanning, web-browser traffic, etc.
2) Wake-up a sleeping device if necessary e.g., packet related to a remote access protocol/application (i.e., accessing office PC from home and vice versa). Meanwhile, NCP also buffers such packets to avoid any packet loss (due to wake-up delay of client device) and replays them to client device after it has woken up. NCP also wakes-up a sleeping device if a proxied application has certain event e.g., new email or message received, VoIP call received, etc.
3) Respond to the packet if related to basic networking protocols (e.g., ARP, PING, DHCP, TCP keep-alive, etc) or proxiable applications (e.g., VoIP and IM clients, etc).

The NCP can be of three different types based on its deployment: (i) on-board NCP [2], (ii) gateway NCP [3] and (iii) standalone NCP [7]. On-board NCP has proxying functionalities embedded into NIC of the actual device. Gateway NCP has proxying functionalities implemented on switch/router whereas standalone NCP is a dedicated PC with NCP functionalities. On-board NCP can proxy a single device whereas, gateway and standalone NCPs can proxy many devices. On-board NCP in [2] is implemented using a USB-based network interface (named Somniloquy). Somniloquy is a pocket size device (200 MHz PXA255-XScale processor, 64 MB RAM) that is attached to PC's USB port. It has 11x to 24x less power consumption than PC in idle state and impersonates PC's presence during sleep periods. Somniloquy can proxy basic network protocols and open-source applications using stubs. The stub is a light-weight version of original application re-written from its source code. Gateway NCP in [3] is implemented on EASY-XWAY-VRX288 Lantiq gateway device (ARM-VR9 processor, 64 MB RAM). It uses a generalized heartbeat message template to proxy applications. Applications need to be modified to include a feature that fills heartbeat message template before sleeping. The approach is future oriented and was tested for an open-source IM application. Standalone NCP in [7] uses a PC (quad-core Intel-XEON-5550 processor, 32GB RAM) that can proxy devices in local network. It uses a dedicated XEN virtual machine for each PC to be proxied. It also uses stubs to proxy applications as in Somniloquy. To achieve energy savings, standalone NCP must proxy several client devices. Thus, energy savings depend on NCP deployment and number of its client devices. Authors in [8] estimated energy savings for different NCP deployments with increasing number of client devices. It has been estimated that NCP has potential to provide billions of dollar savings annually if adopted worldwide.

Although NCP concept is very promising in achieving energy savings, a full-fledged NCP prototype could not be developed until now that can proxy every kind of application including closed-source. Section II addresses challenges which prohibit the development of a full-fledged NCP prototype. This paper also proposes an alternative approach that overcomes the limitations of NCP concept and is also practically realizable.

The proposed system uses a gateway proxy and intelligent collaboration among a user daily used devices (smartphone, tablet, laptop or office PC). The proposed system is addressed in Section III.

II. Challenges in NCP Concept

The NCP concept has been quite successful in proxying basic networking protocols (such as ARP, PING, DHCP, etc), however, proposed strategies in literature for proxying applications are either impractical for today's proprietary applications or limited to only open-source applications. This section addresses key challenges which prohibited the development of a full-fledged functional NCP prototype until now.

A. Proprietary Closed-Source Applications

As addressed in Section I, almost every application periodically sends and receives its specific heartbeat/presence messages. To implement proxying feature for heartbeat messages of an application, the NCP needs the original source code of that specific application. Writing application stub is the most common approach to proxy an application but it is possible only for open-source applications [2], [7]. Authors in [5] proposed application-specific routines on NCP client devices. They continuously monitor application heartbeat messages from the network interface. Before going into sleep state, the last exchanged heartbeat message is provided to the NCP. The NCP then makes necessary changes to provided heartbeat message based on the application in order to proxy it on behalf of sleeping devices. This approach is presented in a more generalized way in [4] by proposing a generalized heartbeat message template. Original applications need to be modified to include a feature that fills heartbeat message template. This strategy is only valid for applications whose payloads vary in predictable fashion e.g., payload contains counter, random value or time-stamp. It is worth to mention that payloads for most applications are very complex and vary in non-predictable fashion. In short, NCP still lacks to proxy proprietary closed-source applications (such as WhatsApp, Viber, Skype, etc) due to non-availability of their source code.

B. Encrypted Packets

Today, almost every application uses encryption and cryptographic signatures. To proxy an application, NCP needs to know security mechanism used by that specific application. NCP also needs to know encryption algorithms, signature algorithms as well as security credentials (such as encryption key, signature key, keys validity, etc). Application developers normally don't share such information. It is worth to mention that some applications (e.g., WhatsApp) uses dynamic security where security credentials have certain validity and replaced/changed periodically (it is possible that every new message has different security credentials applied on it). Thus, it is very challenging for NCP to know what security mechanisms and security credentials are used by an application. It becomes almost impossible for NCP to proxy a proprietary closed-source application that has encrypted packets.

C. TCP Session Maintenance

Preserving open TCP sessions on behalf of sleeping devices is also a challenging task for NCP. Authors in [9] proposed a green TCP/IP concept by making changes to standard TCP/IP. A new field 'Connection Sleep' is included in TCP header which notifies remote peer about changes in device's power state. When the 'Connection Sleep' option is set, remote peer freezes TCP operations and does not send any packet. Another TCP proxying strategy is based on splitting connection [10]. A new shim layer is introduced between socket and application layer. Shim layer is a thin layer that presents socket interface to application with the objective of keeping original socket and application unmodified. It communicates with the shim layer on remote peer for freezing and resuming a TCP session. A Srelay SOCKS service in proposed in [1] as a strategy to proxy TCP sessions. It relays every TCP session between communicating devices, hence faces performance and scalability issues. Authors in [4] proposed TCP session migration strategy. The TCP state is frozen on NCP client device before sleeping and transferred to the NCP for proxying. The updated TCP state is returned back once the NCP client device wakes up. TCP session migration uses a new feature in latest versions of Linux kernels that allows to put a TCP socket in repair state and manipulates it. TCP session migration feature is limited to only Linux OS at present and also requires changes to original applications to include TCP freeze and resume features. It can be observed that all TCP proxying strategies are either facing performance and scalability issues or propose changes to original applications or standard TCP/IP.

D. Mobility Managements

It is very challenging for NCP to proxy mobile devices [11], [12]. The NCP needs to track location of mobile devices and their changing IP addresses. Further, NAT/Firewall will prevent NCP from waking up a mobile client device.

III. How NCP challenges can be addressed?

To overcome NCP limitations (addressed in Section II), this section proposes a new system that is practically realizable with achievable energy savings equivalent to a full-fledged NCP. Since, NCP lacks proxying of modern proprietary closed-source applications, so why not run them on smartphone (apparently appears as a user's smartphone performs application proxying)? For example, when a user puts his office PC into sleep state (i.e., when he leaves his office), the state of applications seamlessly and automatically transfers from his office PC to his smartphone. When user switches ON his home PC, the state of applications seamlessly and automatically transfers from his smartphone to home PC. This strategy will simplify proxying tasks for NCP i.e., only network presence needs to be proxied but not the applications. Instead, proposed system schedules applications execution on different user devices based on their priorities or user specified settings (instead of proxying them). Thus, applications will run on either a smartphone or tablet or laptop or desktop PC at any given time instant. This effectively resolves the main limitations of NCP addressed in Section II.

The proposed system also needs a light-weight proxy on the gateway device that impersonates basic networking protocols (e.g., ARP, PING) on behalf of sleeping devices and wakes them in case of new connection attempts (e.g., remote access). In short, the proposed system is equally beneficial for fixed as well as mobile devices and does not require proxying of TCP sessions and proprietary closed-source applications. A significant portion of battery on an Internet-connected smartphone/tablet is consumed by background applications (they continuously utilize CPU, memory, WiFi/3G/4G interfaces, etc) [12]. In proposed system, battery life of mobile devices will significantly improve due to running applications only for short time when no other device is running them.

An important question that arises is whether a PC application can be supported on smartphone and vice versa. Today, people concurrently use multiple devices in daily life (such as smartphone, tablet, laptop, office PC, etc) which have similar features and capabilities. Modern smartphones and tablets are equipped with quad-core processors, more than 4 GB RAM and 128 GB storage. Further, today they run same Operating Systems (OS) which were initially developed for desktop PCs such as Microsoft Windows 10 and Ubuntu. Apple company also over the time improved integration between its iOS and MAC OS devices which allows task (e.g., writing an email) switching from MacBook to iPhone/iPad and vice versa. Windows, Ubuntu and iOS are breakthrough for smartphones/tablets enabling them to run almost all the same applications as desktop PCs such as email clients (e.g., MS Outlook), IM applications (e.g., Skype, WhatsApp, Viber, etc) and even remote access applications (e.g., SSH, Telnet, etc especially on Ubuntu smartphones). Android OS for smartphones/tablets still has limited flexibility. However, it is worth to mention that even application developers usually implement same application for different operating systems and devices (e.g., Skype, WhatsApp, Viber, Imo, etc are available for smartphones, tablets, laptops and desktop PCs). In short, modern smartphones and tablets can run all the same applications and are normally connected to the Internet 24/7.

From functional point-of-view, the proposed system consists of (i) a gateway proxy for proxying basic networking protocols and waking client devices up whenever necessary (addressed in Section III-A) and (ii) a mechanism for running applications on a single user device at any given time instant (addressed in Section III-B). Section III-C addresses a communication framework based on Universal Plug & Play (UPnP) which ensures automatic discovery and seamless communication among devices in proposed system. Section III-D addresses security and privacy concerns.

A. Basic Architecture of Gateway Proxy

The basic architectural design of gateway proxy is presented in Fig. 2. It is somehow similar to the NCP [4] but with very basic practically realizable features. It consists of five main components: *(i)* network presence proxying rules, *(ii)* wake-up policies, *(iii)* packet filters, *(iv)* packet processor and *(v)* network sockets. Rules in Fig. 2 specify the actions that gateway proxy performs on behalf of sleeping devices.

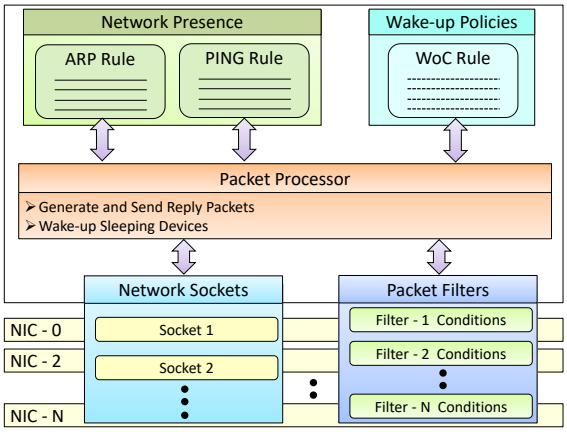

Figure 2. Conceptual architecture of gateway proxy.

Network presence proxying rules consist of ARP and PING. A DHCP rule could also be implemented if some devices use dynamic DHCP-based IP addresses. Wake-up policies contain a Wake-on-Connection (WoC) rule (originally specified in [4]). WoC rule wakes up a sleeping device on receiving a packet on specified transport protocol (TCP or UDP) and port number. It is used for remote access applications such as SSH (TCP port 22), Telnet (TCP port 23), PCAnywhere (TCP/UDP port 5632), RealVNC (TCP port 5900), Remote-Desktop (TCP/UDP port 3389) and others. Packet filters correspond to registered rules (i.e., ARP, PING and WoC) by sleeping devices. They filter packets based on packet header fields (e.g., protocol, port number, etc). A separate filter can be created for each registered rule (note: if a sleeping device supports several remote access applications, a WoC rule should be set for each of them). Packet filters are responsible to hijack packets intended for sleeping devices by using traffic diversion based on ARP spoofing. Packet processor determines the appropriate action for each received packet based on registered rules. Network sockets are used by UPnP communication framework, sending wake-up packets and proxying ARP and PING protocols. The UPnP communication framework is used for communication between the gateway proxy and its client devices (addressed in Section III-C).

B. Adaptive Applications Controlling Service

This service is based on intelligent collaboration among daily used network devices to run applications on only and only one device at any given time instant. This can be achieved in two possible ways: (i) ad-hoc collaboration between devices and (ii) centralized Applications Coordinating/Controlling Unit (ACU).

1) Ad-Hoc Collaboration Between Devices: The devices communicate with one another in an ad-hoc fashion and exchange the list of registered applications, devices priorities and user settings. The basic scenario is depicted in Fig. 3 where a user has four different devices: desktop PC at home, desktop PC at office and two mobile devices (laptop and smartphone). All devices communicate with each other using UPnP communication framework and mutually decide that

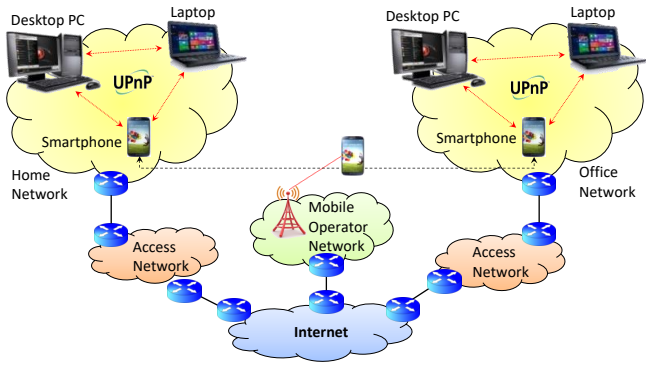

Figure 3. The proposed Ad-Hoc collaborative proxying system.

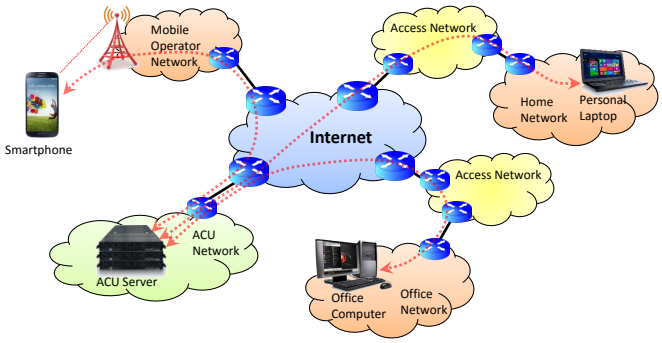

Figure 4. The proposed centralized applications coordinating/controlling service.

which device will run the applications. Although, Ubuntu and Windows smartphones might run almost all the same applications as desktop PCs, the flexibility could be limited for certain smartphones e.g., Android. Still most commonly used applications such as Skype, Viber, WhatsApp, etc are available for different devices and operating systems. In short, proposed system is based on the assumption that applications are supported on different devices (if any smartphone application is not supported on desktop PC or vice versa, such application should be excluded). The proposed system also assumes that as soon as smartphone in Fig. 3 enters home or office network, it automatically disconnects its 3G/4G and connects to WiFi network (due to lower power consumption of WiFi, this feature is already supported by almost all modern smartphones).

The UPnP plays very important role in ad-hoc collaboration and enables devices to automatically discover one another without needing any configurations, network settings or user input. The UPnP makes the whole process seamless and automatic. As soon as the highest priority device goes to sleep state, the applications automatically start-up on the second high priority device. E.g., when a user switches ON computer in home or office, his smartphone automatically discovers it and stops running applications. Meanwhile, applications will run on computer as long as it is in active state. To make the process automatic and seamless, a kernel module can also be developed for computers to track changes in their power state. When the computer goes to sleep state, the kernel module will detect it and communicate with smartphone to start-up the applications. The time when computer is sleeping, the gateway proxy will impersonate basic networking protocols on its behalf and wake it up whenever necessary (e.g., remote access). The ad-hoc collaboration among devices provides flexibility to dynamically register or de-register applications, schedule sleep periods for different devices, wake-up the sleeping devices at exact time and control applications based on priorities or user requests.

2) Centralized Applications Coordinating/Controlling Unit: Unlike ad-hoc scenario, the centralized scenario has decision making and applications controlling service located on a central server. The basic scenario is depicted in Fig. 4. All user devices are equipped with ACU client and communicate with ACU server. The ACU server is a global entity located anywhere in the world and manages user accounts. Each user account can have several registered devices. Fig. 4 depicts the scenario for a single user having three different devices. Based on devices priorities or user-specified settings, the ACU server instructs a single user device to run applications while instructing all others to stop the applications.

C. UPnP Communication Framework

An important design goal of proposed system is to provide autonomous, zero-configuration and seamless communication among devices without requiring any network configurations. This can be achieved using UPnP technology. The UPnP technology is used for (i) communication between user devices and gateway proxy, and (ii) communication among user devices themselves (i.e., considering ad-hoc collaboration among devices). The basic design of UPnP-based communication framework is depicted in Fig. 5. The UPnP Control Point (CP) invokes a specific action on the UPnP Controlled Device (CD) and receives a response back. For two way communication, all devices implement CP as well as CD. The computers implement two types of services: Low Power (LP) and Applications Controlling (AC). The LP service is responsible to manage power status of device e.g., define sleep duration, go to sleep for specified period, etc. While the AC service is responsible for stopping or starting applications based on device priority.

D. Security Considerations

Compared to the NCP concept, proposed system has far less security concerns. The NCP is a public entity and requires user login credentials as well as application-specific security credentials in order to proxy an application. From privacy point of view, a user will be highly satisfied in proposed system as applications presence is maintained on his own devices and privacies are protected. However, communication between user devices and gateway proxy in proposed system is based on UPnP which lacks security features. Thus, communication security is essential to prevent malicious intruders from hijacking and modifying packets in transit. Compared to centralized approach (in Fig. 4), all communications in ad-hoc approach (in Fig. 3) take place inside local network and therefore, face less security concerns. User devices must implement an authentication mechanism (based on certificates or shared

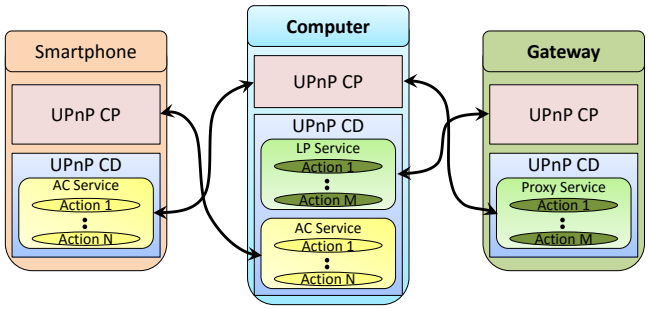

Figure 5. The design of UPnP communication framework.

secret keys) to prevent un-authorized devices or malicious intruders communicating with them. To ensure communication confidentiality and integrity, encryption and cryptographic signatures must be used. Furthermore, it is recommended to use dynamic security policies (i.e., periodically replacing secret keys) for strong protection against cryptanalysis.

IV. Conclusions

The original NCP concept was introduced in literature with two objectives: (i) impersonate basic network presence (i.e., ARP, PING) and maintain reachability (i.e., WoC), and (ii) impersonate Internet-based applications and their associated TCP connections. The first objective faces no serious issues and is practically realizable. The second objective is very challenging that prohibits the development of a full-fledged NCP prototype (i.e., proxying proprietary closed-source applications).

To overcome the limitations of NCP concept, this paper proposed a system which is practically realizable. The proposed system uses a very light-weight gateway proxy that impersonates ARP and PING protocols on behalf of sleeping devices and wakes them up whenever necessary (e.g., remote access). Instead of proxying applications and TCP sessions, the proposed system ensures to run applications on a single user device at any given time instant. Thus, the applications will run on a smartphone or tablet or laptop or desktop PC based on user specified configurations and devices priorities. Not only the proposed system is practically realizable, the achievable energy savings are equivalent or higher than the original NCP concept. Further, it is also beneficial for mobile devices by improving their battery life due to not running the applications 24/7.

References

[1] M. Jimeno, K. Christensen, and B. Nordman, "A Network Connection Proxy to Enable Hosts to Sleep and Save Energy," in *IEEE IPCCC*, 2008.

[2] Y. Agarwal *et al.*, "Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage," in *6th USENIX proceedings*, 2009.

[3] R. Bolla, M. Chiappero, R. Khan, and M. Repetto, "Saving Energy by Delegating Network Activity to Home Gateways," *IEEE Transactions on Consumer Electronics*, 2015.

[4] R. Bolla, M. Giribaldi, R. Khan, and M. Repetto, "Network Connectivity Proxy: Architecture, Implementation and Performance Analysis," *IEEE Systems Journal*, 2015.

[5] R. Khan and S. U. Khan, "Achieving Energy Saving through Proxying Applications on behalf of Idle Devices," in *Procedia Computer Science*, 2016.

[6] P. Werstein and W. Vossen, "A low-power proxy to allow unattended jabber clients to sleep," in *IEEE PDCAT*, 2008.

[7] Y. Agarwal, S. Savage, and R. Gupta, "SleepServer: A Software-only Approach for Reducing the Energy Consumption of PCs within Enterprise Environments," in *USENIX proceedings*, 2010.

[8] R. Bolla, R. Khan, and M. Repetto, "Assessing the Potential for Saving Energy by Impersonating Idle Networked Devices," *IEEE Journal on Selected Areas in Communications*, 2016.

[9] L. Irish and K. J. Christensen, "A Green TCP/IP to Reduce Electricity Consumed by Computers," in *Southeastcon proceedings*, 1998.

[10] C. Gunaratne *et al.*, "Managing Energy Consumption Costs in Desktop PCs and LAN Switches with Proxying, Split TCP Connections, and Scaling of Link Speed," *Network Management*, 2005.

[11] R. Bolla, M. Giribaldi, R. Khan, and M. Repetto, "Smart Proxying: An Optimal Strategy for Improving Battery Life of Mobile Devices," in *Green Computing Conference (IGCC)*, 2013.

[12] R. Bolla, R. Khan, X. Parra, and M. Repetto, "Improving Smartphones Battery Life by Reducing Energy Waste of Background Applications," in *8th NGMAST proceedings*, 2014.

Rafiullah Khan received his Ph.D. degree jointly from University of Genoa, Italy and Polytechnic University of Catalonia, Spain in 2014. He is currently a Postdoctoral Research Fellow at Queen's University Belfast in United Kingdom. His research interests include Ad Hoc Networking, Self-Organizing Networks, Green Networking and Cyber Security.

Sarmad Ullah Khan received his PhD degree from Politecnico di Torino, Italy in 2013. He is currently an Assistant Professor in CECOS university, Peshawar, Pakistan. His research interests include security in wireless sensor networks, Internet of Things, intelligent transportation system and content centric networking.